Un generador matricial de claves frente a Blum Blum Shub.

Rafael Álvarez[1], Joan-Josep Climent[2], Leandro Tortosa[3] y Antonio Zamora[4]

Departament de Ciència de la Computació i Intel·ligència Artificial. Universitat d'Alacant, Campus de Sant Vicent, Ap.Correus 99, E-03080, Alacant, Spain.
afa@afaland.com[1] jcliment@dccia.ua.es[2] tortosa@dccia.ua.es[3] zamora@dccia.ua.es[4]

La enorme popularidad de Internet ha hecho que la demanda de sistemas seguros haya aumentado de forma considerable. Nuestra propuesta es un generador pseudoaleatorio muy eficiente con buenas propiedades criptográficas y, por lo tanto, grandes aplicaciones en estos sistemas seguros. Está basado en las potencias de las matrices triangulares superiores por bloques y, además de lograr resultados estadísticos muy buenos y una gran eficiencia, es un algoritmo muy flexible que se puede adaptar fácilmente a diversas aplicaciones.

1. Introducción

Durante los últimos años, Internet se ha hecho extremadamente popular. Todo el mundo parece estar presente de una forma u otra: las empresas establecen sistemas de comercio electrónico, los gobiernos proporcionan servicios a los ciudadanos, e incluso particulares crean sitios web de las materias más diversas.
A pesar de las inmensas posibilidades, Internet sigue siendo vulnerable a ataques de carácter malicioso. Los dañinos resultados de estos ataques se ven multiplicados por el hecho de que un ataque no implica exclusivamente una interrupción de servicio, una pérdida de dinero o una molestia para el usuario final; también implica un daño a la reputación de la empresa que no muchos se pueden permitir. Las empresas han evaluado esta situación, incrementando la demanda de sistemas y servicios cada vez más seguros. Todos estos sistemas seguros utilizan herramientas criptográficas, generalmente combinando criptosistemas de clave pública (para acordar una clave de sesión) y criptosistemas de clave privada para transmitir datos de forma confidencial.
La mayoría de los sistemas criptográficos se basan en cantidades impredecibles (véanse [13, 16, 17]). Las claves, los números primos o los valores de desafío de muchos criptosistemas necesitan ser lo suficientemente impredecibles para que las probabilidades de todos los valores posibles sean razonablemente las mismas, evitando reducciones del espacio de búsqueda a los valores más probables. Estos valores se obtienen de secuencias aleatorias que pueden ser realmente aleatorias o pseudoaleatorias.
Un generador realmente aleatorio se basa en una fuente natural de aleatoriedad, como el ruido térmico de una resistencia, el ruido captado por un micrófono, el tiempo de emisión de partículas en procesos radioactivos, estadísticas del sistema operativo, etc. Esta fuente es procesada para evitar sesgos y otros defectos que pudiera tener; además, el sistema debe ser diseñado para evitar observación o manipulación por parte de un atacante, comprobando periódicamente que el funcionamiento sea correcto [13, 16, 17].
Un generador pseudoaleatorio [6, 8, 9, 10, 13, 16, 17] es un algoritmo completamente determinista (ya que la secuencia que genera es una función de sus entradas) cuya salida, al contrario que en un generador realmente aleatorio, se puede reproducir. De esta forma, únicamente se necesita la semilla (entrada al generador pseudoaleatorio) para poder reconstruir la secuencia de salida completa. Dicha secuencia de salida es mucho más larga que la semilla y, a pesar de no ser realmente aleatoria, resulta indistinguible de una secuencia aleatoria en la práctica.
Cuando se trata de aplicaciones a la seguridad, necesitamos generar secuencias de grandes períodos, complejidades lineales altas y buenas propiedades estadísticas. Se aplican diversos tests estadísticos a modo de comprobación. Éstos analizan la frecuencia

de bits individuales, parejas y otros patrones de bits; así como la autocorrelación y la complejidad lineal [4, 13, 16, 17].
La mayoría de los generadores criptográficos están basados en registros de desplazamiento con retroalimentación lineal (LFSRs, véanse [5, 7, 15]). Los LFSRs son muy populares porque se pueden implementar fácilmente en hardware y puede analizarse su estructura con relativa facilidad.
Otro generador muy conocido es Blum Blum Shub (BBS) [3]. Se basa en dos números primos de gran tamaño *p* y *q*, que satisfacen $p \equiv 3 \bmod 4$ y $q \equiv 3 \bmod 4$, y un número escogido aleatoriamente *s* que es primo con $n = pq$. Definiendo $X_0 = s^2 \bmod n$ y $X_i = (X_{i-1} - 1)^2 \bmod n$, la secuencia de salida se obtiene tomando el bit menos significativo de cada X_i. La seguridad del generador BBS se fundamenta en el coste computacional que supone factorizar *n*.
Otros generadores combinan un cifrador en bloque de diferentes maneras para obtener una secuencia pseudoaleatoria con seguridad criptográfica, como el generador especificado en ANSI X9.17 que realiza tres cifrados de triple DES en cada iteración [18].
Nuestra propuesta es un nuevo generador basado en técnicas matriciales. Se puede usar para generar claves de sesión, valores de desafío, números de secuencia, etc. Es un algoritmo muy flexible, que se puede ajustar para satisfacer diversos requerimientos de memoria o velocidad. El generador presenta muy buenas características estadísticas y resulta muy eficiente frente a BBS.
El resto del artículo está organizado de la siguiente manera: en la sección 2 se recuerdan algunos conceptos básicos de algebra lineal; el generador se describe con detalle en la sección 3; la sección 4 incluye los resultados experimentales obtenidos para nuestra propuesta, comparándolos con BBS; finalmente, exponemos algunas conclusiones en la sección 5.

2. Preliminares

En esta sección, presentamos algunas nociones y propiedades básicas del álgebra lineal que son necesarias para nuestra propuesta (véase [11] para una descripción más detallada). Además, introducimos la notación matemática que vamos a usar posteriormente.
Sea

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + x^n$$

un polinomio mónico en $Z_p[x]$. La matriz asociada a f es la matriz $n \times n$

$$\overline{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix}$$

Si f es irreducible en $Z_p[x]$, entonces el orden de $\overline{A}$ es igual al orden de cualquier raíz en f en F_{p^n} y el orden de $\overline{A}$ divide a $p^n - 1$. Además, si asumimos que f es un polinomio primitivo en $Z_p[x]$, entonces el orden de $\overline{A}$ es exactamente $p^n - 1$.
En consecuencia, si trabajamos con polinomios primitivos en $Z_p[x]$, podemos construir fácilmente matrices cuyo orden sea máximo.
Odoni, Varadharajan y Sanders [14], proponen un esquema extendido que está basado en la construcción de la matriz por bloques

$$\overline{A} = \begin{bmatrix} \overline{A}_1 & 0 & \cdots & 0 \\ 0 & \overline{A}_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \overline{A}_k \end{bmatrix},$$

donde $\overline{A}_i$ es la matriz asociada a f_i, siendo f_i, para $i = 1, 2, \cdots, k$, polinomios primitivos distintos en $Z_p[x]$ de grado n_i, para $i = 1, 2, \cdots, k$ respectivamente. El orden de $\overline{A}$ es $p^{n_i} - 1$, para $i = 1, 2, \cdots, k$, por tanto, el orden de $\overline{A}$ viene dado por $mcm(p^{n_1} - 1, p^{n_2} - 1, \cdots, p^{n_k} - 1)$.
Con el objetivo de usar una matriz de este tipo en un sistema de clave pública, los citados autores conjugan esta matriz $\overline{A}$ con una matriz invertible P, de tamaño $n \times n$, con $n = n_1 + n_2 + \cdots + n_k$, obteniendo una nueva matriz $A = P\overline{A}P^{-1}$ que tiene el mismo orden que $\overline{A}$.

3. Descripción del generador

Nuestro generador está basado en las potencias de las matrices triangulares superiores por bloques definidas en Z_p, con p primo. A medida que se van calculando las diferentes potencias de estas matrices obtenemos, como resultado, una secuencia de matrices con un período muy grande y con muy buenas propiedades de aleatoriedad. Cada elemento de la secuencia se puede procesar para obtener una serie de valores que conformen una secuencia de salida con buenos resultados estadísticos. Este esquema es lo suficientemente simple para ser muy rápido pero incorpora la suficiente complejidad para tener buenas características criptográficas; además, se puede adaptar para crear otras primitivas criptográficas.
Considérese la matriz triangular superior por bloques M definida como

$$M = \begin{bmatrix} A & X \\ O & B \end{bmatrix}, \qquad (1)$$

cuyos elementos pertenecen a Z_p, donde A es una matriz $r \times r$, B es una matriz $s \times s$, X es una matriz $r \times s$ y O denota la matriz nula de tamaño $s \times r$.
El siguiente resultado, base de nuestro generador, establece la expresión de las diferentes potencias de la matriz M y define la matriz $X^{(h)}$ en función de A, B y X.

Teorema 1 Sea M la matriz triangular superior por bloques dada en (1). Para cualquier entero no negativo, h, se verifica

$$M^h = \begin{bmatrix} A^h & X^{(h)} \\ O & B^h \end{bmatrix}, \quad (2)$$

donde

$$X^{(h)} = \begin{cases} 0 & si\ h = 0, \\ \sum_{i=1}^{h} A^{h-i} X B^{i-1} & si\ h \geq 1. \end{cases} \quad (3)$$

Además, si $0 \leq t \leq h$ entonces

$$X^{(h)} = A^t X^{(h-t)} + X^{(t)} B^{h-t}. \quad (4)$$

Demostración. Se demuestra en primer lugar la ecuación (2), usando inducción sobre *h*.
Para $h = 0$ y $h = 1$ el resultado es obvio.
Suponiendo cierta la ecuación (2) para $h-1$, se comprobará que lo sigue siendo para *h*.
Claramente,

$$\begin{aligned} M^h &= MM^{h-1} \\ &= \begin{bmatrix} A & X \\ O & B \end{bmatrix} \begin{bmatrix} A^{h-1} & X^{(h-1)} \\ O & B^{h-1} \end{bmatrix} \\ &= \begin{bmatrix} A^h & AX^{(h-1)} + XB^{h-1} \\ & B^h \end{bmatrix}. \end{aligned}$$

De la hipótesis de inducción y de (3) se tiene

$$\begin{aligned} AX^{(h-1)} + XB^{h-1} &= A\sum_{i=1}^{h-1} A^{h-1-i} X B^{i-1} + XB^{h-1} \\ &= \sum_{i=1}^{h-1} A^{h-i} X B^{i-1} + XB^{h-1} \\ &= A\sum_{i=1}^{h} A^{h-i} X B^{i-1} \\ &= X^{(h)}; \end{aligned}$$

obteniéndose la misma expresión que en (2).
También, si $0 \leq t \leq h$, se tiene

$$\begin{aligned} M^h &= M^t M^{h-t} \\ &= \begin{bmatrix} A^t & X^{(t)} \\ O & B^t \end{bmatrix} \begin{bmatrix} A^{h-t} & X^{(h-t)} \\ O & B^{h-t} \end{bmatrix} \\ &= \begin{bmatrix} A^h & A^t X^{(h-t)} + XB^{h-t} \\ & B^h \end{bmatrix}. \end{aligned}$$

Comparando este resultado con (2) se obtiene (4).

Para generar la secuencia pseudoaleatoria de bits, se fijan las matrices *A* y *B* utilizando polinomios primitivos y se elige de forma aleatoria la matriz *X*, que constituye la semilla de la secuencia. A continuación se aplica la expresión (4) para obtener la secuencia de matrices siguiente:

$$X^2, X^3, X^4, \ldots \qquad (5)$$

Para cada matriz, se establece una operación de extracción de bits que consiste en la suma de todos los elementos de $X^{(h)}$, obteniendo un nuevo elemento $x^{(h)}$, $h = 2, 3, 4, \ldots$, en Z_p del cual extraemos el bit menos significativo, $b^{(h)}$, de su expresión binaria. De esta forma, se obtiene la secuencia de bits

$$b^2, b^3, b^4, \ldots$$

Alternativamente, se puede modificar este esquema de extracción tomando tantos bits por iteración como sea necesario o realizando operaciones más complejas para adaptar el algoritmo a otras necesidades.
Esta secuencia se filtra posteriormente, mejorando la seguridad y eliminando el sesgo de la secuencia, con el proceso siguiente:

$$c^{(i)} = b^{(i)} \oplus c^{(i-1)} \ \ para\ i = 2, 3, 4, \ldots \quad y\ c^{(1)} = 0. \qquad (6)$$

A continuación se analiza la forma de obtener períodos grandes para la secuencia dada en (6).
La clave para la solución de este problema se ha adelantado en la sección 2.
Sean

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{r-1} x^{r-1} + x^r$$

y

$$g(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{s-1} x^{s-1} + x^s$$

dos polinomios primitivos en Z_p; y $\overline{A}$ y $\overline{B}$ las matrices asociadas correspondientes. Sean *P* y *Q* dos matrices invertibles tales que $A = P\overline{A}P^{-1}$ y $B = Q\overline{B}Q^{-1}$. Con este esquema, el orden de la matriz *M* de (1) es

$$mcm(p^r - 1,\ p^s - 1)$$

El valor de *p* o los tamaños de *A* y *B* no necesitan ser muy grandes para lograr grandes períodos, como se ve en la tabla 1, que muestra algunos resultados para el período en función de los parámetros *p*, *r* y *s*.
El valor que aparece en la columna cifras representa el número de cifras del período de la secuencia binaria (el entero 2^{128} tiene 39 cifras). Obsérvese que los valores tomados para *r* y *s* son primos entre si, con la intención de optimizar el período.

p	r	s	Cifras	p	r	s	Cifras

3	32	31	30	19	16	19	39
	48	47	39		32	31	57
	64	63	47		64	63	98
5	32	31	38	31	16	15	40
	30	33	39		32	31	64
	64	63	61		64	63	111
7	24	27	39	251	12	13	46
	32	31	43		32	31	76
	64	63	70		64	63	168
11	22	21	39	257	9	10	40
	32	31	50		32	31	93
	64	63	67		64	63	169

Tabla 1. Orden de M para diferentes valores de p, r y s.

Para finalizar, se debería recalcar que la generación de la secuencia (6) es muy eficiente si consideramos $t = 1$ en la expresión (4); y que todos los bits de la salida dependen de la semilla *X*.

4. Resultados

4.1 Parámetros de los Tests

El generador ha sido analizado con cinco estadísticas diferentes [2, 13] (tests de frecuencia y autocorrelación) y con el cálculo de la complejidad lineal de la secuencia [1, 12]. En estos tests, *n* es la longitud de la secuencia.
El test monobit es un test de frecuencia de bits individuales, diseñado para comprobar que el número de unos y ceros de la secuencia es aproximadamente el mismo.
Este test queda expresado con la siguiente ecuación, siendo n_0 y n_1 el número de bits cero y uno respectivamente:

$$X_1 = \frac{(n_0 - n_1)^2}{n}.$$

Este test sigue una distribución χ^2 con 1 grado de libertad.
El test serial comprueba que la frecuencia de las parejas de bits (00, 01, 10 y 11) es también, más o menos, la misma para todas. Los parámetros n_{00}, n_{01}, n_{10} *y* n_{11} son el número de ocurrencias de dichas parejas permitiendo el solapamiento.

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1.$$

Se sigue una distribución χ^2 con 2 grados de libertad.
El test de poker comprueba que los patrones de longitud *m* aparecen el mismo número de veces en la secuencia sin permitir el solapamiento. El valor de *m* se obtiene con $\left\lfloor \frac{n}{m} \right\rfloor = 5 \cdot 2^m$, *k* se toma como $k = \left\lfloor \frac{n}{m} \right\rfloor$ y n_i es el número de ocurrencias del patrón *i*.

$$X_3 = \frac{2^m}{3}\left(\sum_{i=1}^{2^m} n_i^2\right) - k\,.$$

Sigue una distribución χ^2 con $2^m - 1$ grados de libertad.
Una racha es un patrón de todo ceros o todo unos, un bloque es una racha de unos y un hueco es una racha de ceros. El número esperado de rachas de longitud *i* es $e_i = (n - i + 3)/2^{i+2}$. El valor de *k* se toma como el mayor entero *i* para el que $e_i \geq 5$ y es el límite de longitud para el que se contabilizan rachas; finalmente, B_i *y* G_i son el número de bloques y huecos de longitud *i* (hasta longitud *k*) de la secuencia. El test de rachas se expresa con:

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i};$$

siguiendo una distribución χ^2 con $2k - 2$ grados de libertad.
La autocorrelación comprueba las coincidencias entre la secuencia y su desplazada *d* bits. Tomando $A(d)$ como la cantidad de bits distintos entre la secuencia y su desplazada *d* bits, tenemos:

$$X_5 = \frac{2\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}};$$

que sigue una distribución N(0,1).
La complejidad lineal de una secuencia de bits es la longitud del LFSR más pequeño que pueda generar dicha secuencia. Si dibujamos la complejidad lineal de una secuencia frente a su longitud (para $n = 1, 2, 3, \ldots$) tenemos el perfil de complejidad lineal, que debería seguir de forma ajustada la recta $n/2$ si la secuencia es aleatoria. La complejidad lineal de una secuencia se puede calcular con el algoritmo de Berlekamp-Massey [1, 12].

4.2 Resultados de los Tests

La complejidad lineal esperada para una secuencia aleatoria es $n/2$. Para el resto de los tests, los resultados se comparan con los valores umbrales que se dan en la tabla 2, pasando el test aquellos por debajo del umbral. El nivel de significancia, α, es la probabilidad de que una secuencia falle el test incluso cuando debería haberlo pasado; de esta forma cuanto mayor es el nivel de significancia más restrictivo se es con los resultados.

α	Monobit	Serial	Poker	Rachas				Autocorrelación
				$2\cdot10^3$	$2\cdot10^4$	$2\cdot10^5$	$2\cdot10^6$	
0.001	10.830	13.820	330.5	29.59	39.25	51.18	59.70	3.090
0.005	7.870	10.590	316.9	25.19	34.27	45.56	53.67	2.576
0.010	6.635	9.210	310.5	23.21	32.00	42.98	50.89	2.326

0.025	5.024	7.378	301.1	20.48	28.85	39.36	46.98	1.960
0.050	3.842	5.992	293.2	18.31	26.30	36.42	43.77	1.645
0.100	2.706	4.605	284.3	15.99	23.54	33.20	40.26	1.282

Tabla 2. Valores umbrales para los tests estadísticos.

Durante la extensa experimentación realizada, se han considerado diferentes valores para *p*, *r* y *s*, combinados con un gran número de secuencias de distintas longitudes, obteniendo en todos los casos excelentes resultados para los seis tests descritos en esta sección.

A modo de ejemplo, en la tabla 3 se reflejan los resultados para $p = 257$, $r = 8$ y $s = 9$. Los polinomios primitivos empleados para obtener las matrices *A* y *B*, tal y como se describe en la sección 3, son $f(x) = x^8 + x + 19$ y $g(x) = x^9 + x + 6$ respectivamente. La semilla *X* es una matriz 8×9 obtenida usando un generador aleatorio basado en ruido blanco.

El test de poker se ha realizado sobre secuencias de 8 bits y para el de autocorrelación se han calculado todos los índices de desplazamiento posibles ($d = 1, 2, \ldots, n/2$) quedándose con el peor valor de todos los obtenidos.

Hemos realizado cada test con diferentes secuencias de longitudes $2 \cdot 10^3$, $2 \cdot 10^4$, $2 \cdot 10^5$ y $2 \cdot 10^6$; y con valores de *X* diferentes.

Podemos ver que los resultados son particularmente buenos, mucho mejores que los valores umbrales con $\alpha = 0.100$ en la tabla 2; se aprecia también que son muy regulares, sin valores extraños, y con todos los valores pasando los tests de igual forma.

Según vamos incrementando el número de bits producidos vemos que no hay una relación directa entre la longitud de la secuencia y los valores de los tests, debiéndose las diferencias, en gran medida, a que se han empleado matrices *X* distintas para cada secuencia.

Los tests de complejidad lineal también ofrecen resultados excelentes, obteniendo $n/2$ en los tres casos. Esto significa que la secuencia presenta la misma no linealidad que una secuencia realmente aleatoria, evitando que sea predecible a causa de una alta linealidad. El perfil de la complejidad lineal se muestra en la figura 1, donde se puede apreciar lo bien que se ajusta a la recta $n/2$.

Longitud	Monobit	Serial	Poker	Rachas	Autocorrelación	Complejidad Lineal
$2 \cdot 10^3$	1.352	1.514	251.7	11.260	0.800	1001
$2 \cdot 10^4$	0.012	1.579	243.1	8.441	0.800	10001
$2 \cdot 10^5$	1.095	1.451	239.1	26.550	0.789	100000
$2 \cdot 10^6$	0.814	3.313	207.2	37.590	0.807	1000000

Tabla 3. Resultados de los tests estadísticos.

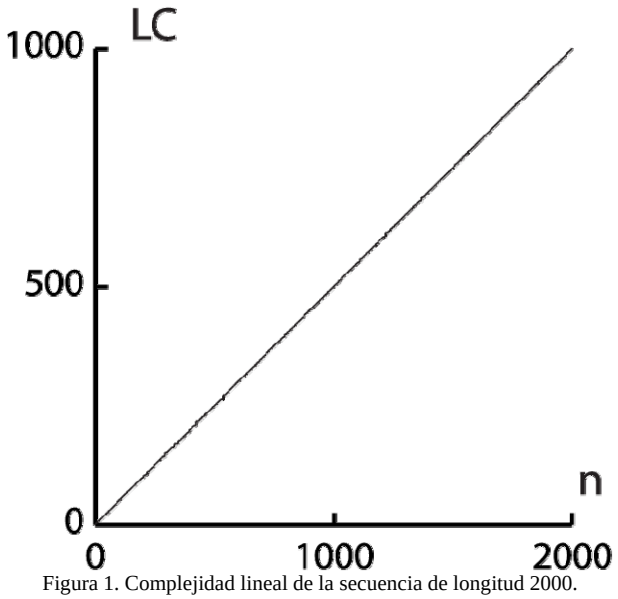

Figura 1. Complejidad lineal de la secuencia de longitud 2000.

4.3 Comparativa con el generador BBS

Mostramos en la tabla 4 una comparativa entre nuestro algoritmo y el conocido generador BBS. El tiempo que se muestra es el consumido por ambos algoritmos para generar una secuencia de $2 \cdot 10^5$ bits; sin tener en cuenta los precálculos o transformaciones a la semilla, sólo la generación de bits.
El test se ha realizado sobre el mismo ordenador, con el mismo compilador y las mismas opciones de optimización para hacer el test tan equilibrado como sea posible. Para secuencias de diferentes longitudes se han obtenido resultados similares.
Estos resultados confirman que el generador propuesto es mucho más rápido que BBS y que los resultados estadísticos son mejores para la mayoría de los parámetros. La principal ventaja de nuestro generador estriba en su flexibilidad, permitiendo elegir tamaños de matriz y esquemas de extracción de bits optimizados para una aplicación en concreto, eligiendo entre velocidades altas, mayor seguridad o utilización de memoria mínima.

Algoritmo	Monobit	Serial	Poker	Rachas	Autocorrelación	CL	Tiempo
Matricial	0.041	0.657	237.5	20.798	0.784	100000	0.013 s
BBS	0.776	1.188	272.9	10.342	0.792	100000	1.020 s

Tabla 4. Comparativa con BBS para una secuencia de 20000 bits.

5 Conclusiones

Con el objetivo de aportar herramientas útiles en los sistemas seguros, hemos presentado un nuevo generador pseudoaleatorio, con grandes aplicaciones criptográficas, basado en las potencias de matrices triangulares superiores por bloques. Es de destacar que, usando polinomios primitivos para generar los bloques diagonales y tomando tamaños de primos y matrices muy pequeños, podemos producir secuencias de grandes períodos, buenas propiedades estadísticas y alta complejidad lineal; de esta forma, la secuencia se genera de forma muy eficiente ya que no requiere el coste computacional que sería necesario con grandes primos o enormes matrices.

Incluso con tamaños tan pequeños, el algoritmo asegura un alto nivel de seguridad debido a que el espacio de claves es muy grande y a que la semilla participa en la generación de cada bit.
El algoritmo resulta todavía más útil si se tiene en cuenta que el tamaño de las matrices y el número de bits que se toman por iteración se pueden elegir de forma precisa para cada aplicación, pudiendo utilizar muy poca memoria o conseguir velocidades muy altas.

Referencias

1. Berlekamp, E. R.: Algebraic Coding Theory. McGraw Hill, New York (1968)
2. Beker, H., Piper, F.: Cipher Systems: The Protection of Communications. John Wiley and Sons, New York (1982)
3. Blum, L., Blum, M., Shub, M.: A Simple Unpredictable Pseudorandom Number Generator. SIAM J. Comput. vol. 15 (1986) 364-383
4. Blaze M., Diffie W., Rivest R., Schneier B., Shimomura T., Thompson E., Weiner M.: Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. Counterpane Labs Publications, (1996)
5. Fuster A., García J.: An Efficient Algorithm to Generate Binary Sequences for Cryptographic Purposes. Theoretical Computer Science, 259 (2001) 679-688
6. García J., Rodríguez M.: A Family of Keystream Generators with Large Linear Complexity. Applied Mathemathics Letters, 14 (2001) 545-547
7. Golomb, S. W.: Shift Register Sequences. Aegean Park Press, California (1982)
8. Kelsey, J., Schneier, B., Ferguson, N.: Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. Sixth Annual Workshop on Selected Areas in Cryptography. Springer Verlag, (1999)
9. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Cryptanalitic Attacks on Pseudorandom Number Generators. Fast Software Encryption, Fifth International Workshop. Springer-Verlag, (1998) 168-188
10. Lagarias, J. C.: Pseudorandom Number Generators in Cryptography and Number Theory. Proc. Symposia in Applied Mathematics, Cryptography and Computational Number Theory. Carl Pomerance Ed. vol. 42 (1990) 115-143
11. Lidl, R., Niederreiter, H.: Introduction to Finite Fields and their Applications. Cambridge University Press, Cambridge (1994)
12. Massey, J. L.: Shift-Register Synthesis and BCH Decoding. IEEE Transactions on Information Theory, 15 (1969) 122-127
13. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Florida (2001)
14. Odoni, R. W. K., Varadharajan, V., Sanders, P. W.: Public Key Distribution in Matrix Rings. Electronic Letters, vol. 20 (1984) 386-387
15. Rueppel, R. A.: Analysis and Design of Stream Ciphers. Springer-Verlag, Berlin (1986)
16. Schneier, B.: Applied Cryptography Second Edition: protocols, algorithms and source code in C. John Wiley and Sons, New York (1996)
17. Stallings, W.: Cryptography and Network Security: Principles and Practice. Third Edition. Prentice Hall, New Jersey (2003)
18. Accredited Standards Committee, X9 - Financial Services: American National Standard, Financial Institution Key Management. X9 - Secretariat, American Bankers Association, ANSI X9.17 (1995)